Towards Procedural Generation As Gameplay: CLAY and Tombs of Tomeria

Michael Cook and Simon Colton
The Metamakers Institute
Games Academy
Falmouth University

ABSTRACT
Procedural generation is a popular tool for generating large quantities of content for games, but its function as a mechanic is largely underexplored. In this paper we describe *CLAY* and *Tombs Of Tomeria*, two games in which the player can reparameterise a level generator as a means of exploration. We describe the motivation for the work, the design challenges in using procedural generation in this way, and discuss future problems and opportunities with using generative techniques as game mechanics.

Keywords
procedural generation, game design

INTRODUCTION
Procedural generation is now a fundamental part of the language of game design, a tool that can be used to challenge players (Mossmouth Games, 2008), create atmosphere (Hello Games, 2016) or spur creativity (Mojang, 2009), as well as simply extending the length of a game. Despite the increasingly commonplace nature of procedural generation in games, it is largely applied for the same ends in most games – usually the production of endless content for the player to encounter, a one-way process in which the user is a passive consumer.

Many other kinds of game technology eventually made the leap from tool to mechanic. Physics engines, for example, first appeared in games adding heightened sense of realism and verisimilitude to game worlds. But they were soon repurposed to become the point of the game - the term 'physics-based game' now describes a large and popular genre of game. In (Treanor et al., 2015) the authors coin the term *AI-based game* and suggest that AI techniques might make a similar transition to become the driving force for a game's design (and show examples of games where this is already the case). Procedural generation has yet to make such a leap in a meaningful way, however.

In this paper we describe the development of two games - *CLAY* and *Tombs of Tomeria* - in which the primary game mechanic involves taking control of a procedural generator. We describe the development process of these games, we evaluate the use of procedural generation as a game mechanic, and we discuss the ongoing development of this idea in both Tombs of Tomeria and future prototypes. The remainder of this paper is organised as follows: in *BACKGROUND* we describe the notion of AI-based game design and discuss games which have historically tried to involve the player in generative systems; in *CLAY* we describe a jam game which was the basis for our work in the area; in *TOMBS OF TOMERIA* we describe a larger follow-up game we are developing that expands on these ideas and tries to adapt them into a more focused game design; in *DISCUSSION* we identify some lessons learned through developing this games and outline areas we hope to develop further.

Proceedings of 1st International Joint Conference of DiGRA and FDG

BACKGROUND

In (Treanor et al., 2015) the authors propose the notion of *AI-based game design*, in which a core part of a game's design is *foregrounded* AI - AI systems that the player is aware of, can reason about, and can interact with. One example of such a game would be *Spy Party*, a multiplayer game in which one player must mimic the actions of other AI agents while a second player tries to distinguish the human avatar from the crowd. Both players must think about the way in which computer-controlled characters act, and how that behaviour differs from how a human might act. Interaction with, and understanding of, AI systems is fundamental to gameplay. The work in (Treanor et al., 2015) heavily influenced the development of these prototypes, as we are in effect *foregrounding* procedural content generators in the game.

Some games frame the exploration of generative space as part of gameplay. In *Inside A Star-Filled Sky* the player can 'enter' any enemy they encounter, and the shape of the enemy dictates the shape of the level they enter, which itself contains enemies. This recursive process of going to higher and lower levels is a way of exploring a vast interconnected space produced by a generator, albeit in a way largely disconnected from the generative process.

Endless Web is the game which most strongly exemplifies the idea of using procedural generation as a game mechanic (Smith et al., 2012). Endless Web is a platformer where each level is a procedurally generated mix of different challenge elements. By taking certain exits from a level, the next level is generated under different parameters, allowing the player to explore the generative space figuratively as well as literally (this exploration ties into an in-game map). Endless Web is a major inspiration for the games we describe here.

CLAY

CLAY is a game prototype developed for the 2015 AI Jam. You play as an astronaut reactivating drones stationed in a hollowed-out planet. You explore the planet, collecting jump packs and reactivating as many drones as possible, before returning to your ship. The player achieves this by using buttons to control the shape and size of the planet's remains. Pressing certain buttons increases the density and quantity of matter in the level, while other buttons reduce it. The player can preview changes before they apply them, allowing them to plan a route through each level to activate drones and safely get back to their ship. Figure 1 shows a screenshot of the game, which can be played for free online[1].

Levels are generated using a cellular automata generator. A level is initially randomly seeded with solid and empty tiles, and then the level generator goes through several iterations of applying rules to the grid of tiles. Depending on the tile's state (either solid or empty) and the number of solid neighbours it has, an iteration may switch a tile state from solid to empty or vice versa. Over time, sparse groups of tiles coalesce into solid walls and organic shapes begin to form. The formation of these solid areas, how large they are, how many individual solid areas they form and so forth are governed by several parameters, the most important of which we describe below.

1. https://cutgarnetgames.itch.io/clay

Figure 1: A screenshot from CLAY.

- **Initial Random Chance** (IRC): When seeding a map initially, how likely is a given tile to start solid? A higher IRC results in a denser map overall.
- **Number Of Iterations** (NOI): The number of times the birth/death rules are applied to the grid. Generally, more iterations results in a smoother level layout.
- **Birth Limit** (BL): If a tile is empty, and it has at least BL solid neighbours, it becomes solid.
- **Death Limit** (DL): If a tile is solid, and it has fewer than DL solid neighbours, it becomes empty.

In CLAY, the player has control over two of these parameters: the initial random chance, and the number of iterations. They can adjust the IRC by 5% at a time, and adjust the number of iterations by 1 at a time. Each time they attempt to make an adjustment, they are shown a preview of what the resulting level will look like, before confirming the change. Figure 2 shows a screenshot of this preview process. The play can then cancel the change, or confirm it, leading to a fade-out while the level regenerates and then places the player back in it. Because the player starts at the top of the level and must also exit from there too, traversing back up the level is an important part of gameplay, and so the limited jump boosts combine with the ability to change the level structure to help the player get back up the level.

Feedback from players of CLAY was generally positive – they enjoyed using the generative controls to explore the level, and moving back and forth between denser and more open states was a satisfying kind of generative puzzle. Some users noted that the slow process of switching between preview states and regenerating the level discouraged them from experimenting more. We also had personal concerns about the game mechanic - firstly, that it gave the player too much control over the level, which made it hard to make navigation a

Figure 2: A screenshot from CLAY showing a preview of a change to the level. Grey areas will be solid after a change, non-grey areas will be empty.

challenge once the concept had been grasped. Second, that it was hard to convey what you were doing when controlling the level design. Even with the sci-fi narrative, it was unclear what it meant to increase the amount of solid space around you (the game contextualises it as 'terraforming' and describes the processes as density and erosion). The mechanic was hard to explain to the player, and as a result it felt artificial.

TOMBS OF TOMERIA

Tombs of Tomeria (henceforth just 'Tombs') is a 2D platformer set in a series of underground tombs, in which the player must navigate through a shifting landscape in search of treasure. Levers and switches are scattered throughout the level, and activating these cause the walls of the tomb to shift and change, revealing new passageways or cutting off existing ones. Figure 3 shows a screenshot from the game, the latest public version of which can be downloaded for free online[2].

Tombs was designed as a successor to the ideas prototyped in CLAY. The objective with Tombs was partly to design a bigger and more developed game using the same principles, but we also changed the way the procedural generator was integrated into the game in order to improve the relationship between the player and the generator-driven mechanics.

The most fundamental change was to remove the player's ability to freely change the level parameters at any time. Instead of a universal power over the generator at all times, the player has to track down specific places where interaction with the world is possible, primarily in the form of levers. This reduces the scope of the player's control over the world, but also focuses it so that changing the level is a more meaningful action. It's also much easier to explain and understand – the player is causing the walls of the tomb to shift in and out by activating machinery. They can see that the walls are always present in the back-

2. http://cutgarnetgames.itch.io/tombs-of-temeria

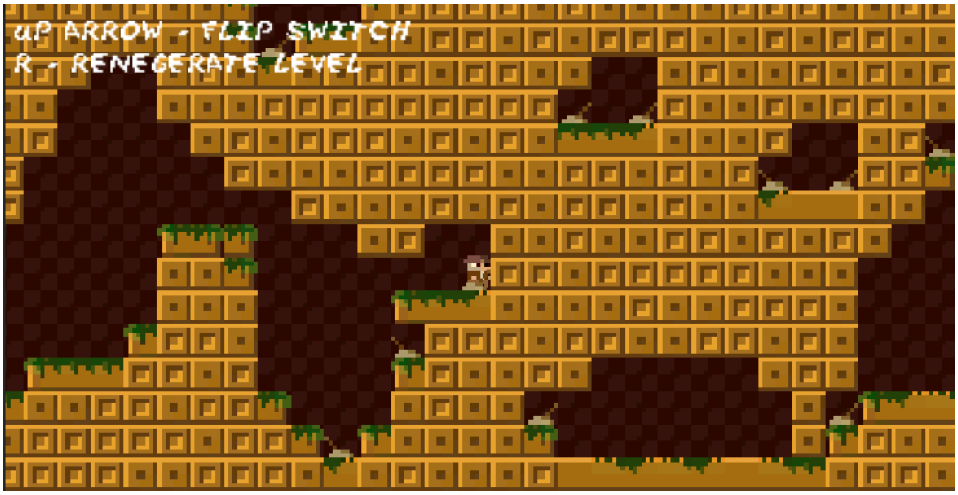

Figure 3: A screenshot from Tombs Of Tomeria. The switches allow the player to change the level.

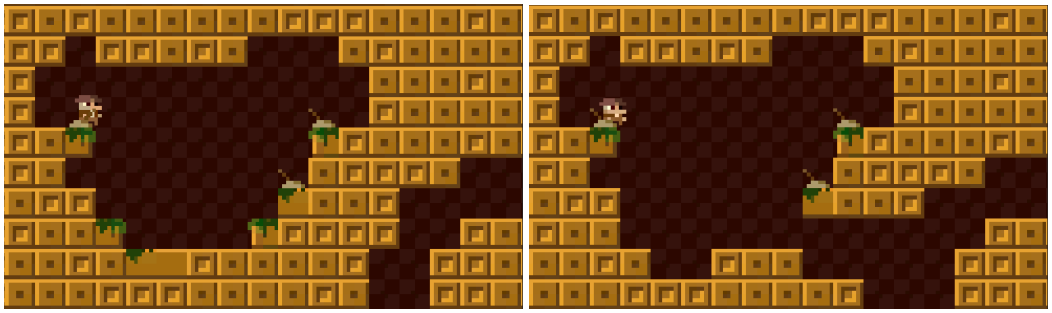

Figure 4: Pulling a lever in Tombs of Tomeria to reveal a new path.

ground, so solid space doesn't disappear and reappear as it did in CLAY, instead it simply recesses itself temporarily, allowing the player to pass by.

Limiting where and how the player can interact with the world also makes this knowledge available to the level designer at runtime, which means that Tombs can calculate at the beginning of a level where the player can go and what levers are accessible to them. While this is theoretically possible in CLAY, the state space is vast - since the level can be altered in any way, at any position, and can conceivably be in a large number of configurations before a change. In Tombs, changes are limited and easily enumerable. This allows Tombs to become a puzzle game in a more literal way than CLAY - if there are four levers in the world, what sequence do the levers need to be pulled in to reach the exit? Because pulling a lever may cut off some areas of the map while enabling access to others, different sequences of levers may be required to access other parts of the map with levers that have more powerful effects. Tombs can calculate these paths before the game begins, evaluating how hard a level is to solve based on how many levers have to be pulled and in what order, or how many potential dead-ends the player might encounter.

In order to calculate these paths, and also to improve the speed with which we respond to changes to the level, we compute all possible states the game level can be in before the game begins. We begin by selecting a random seed for the level, since this is needed to remain consistent when reparameterising the generator. Then we calculate every combination of parameter settings it is possible be in. Since we use fixed steps for each lever pull (the same 5% IRC and 1 iteration differences we used for CLAY) and we define upper and lower bounds on parameter values, this is a finite list. A level in Tombs has 42 different instantiations given the current parameter steps and upper and lower bounds. We discuss the implications of this precomputation step in the later discussion section.

Tombs currently creates levels in a random fashion, which is to say it places the player start, exit and levers in random positions in the game world. It then tests to see if the level is solvable, and if it isn't it restarts with a new seed. To test for solvability we perform an A* search not through the physical space of the level, but through the meta-level space of possible game states. In this meta-space a vertex represents a particular configuration of the game. The information stored for each vertex is a list of levers in the level and their status (switched left or right), and the position of the player. Instead of storing the specific location of the player, we instead store which 'chamber' they are in (a chamber being a self-contained empty space in the level, such as the one the player is in in the first image in Figure 5).

An edge between two vertices in the meta-level graph means that the player can access a lever from their current chamber which transforms the game state into a new state. The A* search to test for solvability simply tests to see if the game state can be transformed from its original configuration, to one in which the player and the exit exist in the same chamber, meaning that the player has managed to reach the exit and complete the level. To avoid trivial levels being generated, we only accept solutions with three or more lever pulls in their solution.

While this process of checking a level is relatively straightforward, it is imperfect. For one thing, the solver assumes perfect play yet it is very easy for the player to get stuck down a crevasse and unable to get out. Spelunky is a notable example of a game which designed around this problem by allowing the player a limited ability to blow up walls and throw ropes up to climb from pits, and we are considering similar techniques for Tombs (perhaps, in keeping with the design, a limited number of explosive charges which reduces or increases one of the parameters at random). The solver can also miscalculate the player's ability to jump to certain areas due to an imperfect representation of the game's physics system, which is another area we are seeking to improve.

DISCUSSION

Development on Tombs is still ongoing, but working on both it and CLAY has led to some initial ideas about the use of procedural generation as a game system which we share here both as points of discussion and ideas that are shaping our future work in the area.

Power

Procedural content generation is often used in games to produce large quantities of critical game content that defines the player's progression through game systems (such as the

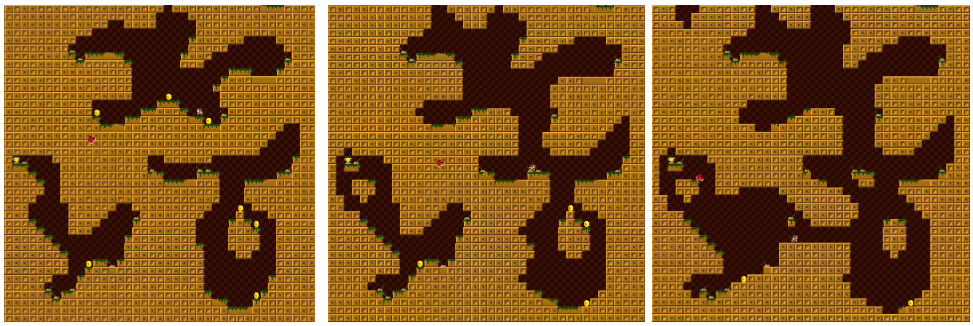

Figure 5: Three stages of a level in Tombs being solved.

generation of items, enemy placements, or environments to explore). Allowing the player control over such systems can in some cases be akin to allowing them control over the very nature of the game's challenge or purpose. CLAY is an example of a game which failed to properly check this, and as such the player was given the ability to trivialise the game if they so wished.

We consider this an interesting result in itself –after all, exploring the generative space of a procedural generator can be a playful experience and some players did report that simply drifting through space and moulding the world was interesting to them. However, this is an important point to bear in mind for designers wishing to produce specific kinds of challenge for the player. Tombs shows how the power of controlling a procedural generator can be limited in interesting ways, and we hope to pursue both the playful and puzzling aspects of this in future work.

Tractability

Procedural generation is typically employed at large scales, with very large generative and parametric spaces involved. When content is produced and used with little filtering or testing, this scale is useful as it provides a sense of unpredictability and an endless source of low-level novelty. In the context of games like CLAY and Tombs, however, the need to reason about the generative space means that the scale of procedural generators must be controllable in some way. In Tombs we set limits on the extent to which parameters could be changed and fixed intervals for how much they could be changed at a time, allowing us to limit the player's influence over the generator to a reasonable space which is amenable to analysis while still being interesting to the player.

As with questions of power, clearly for many kinds of design tractability will not be a factor. The unpredictability of the player's control may factor into a playful or creative interaction with procedural systems, for example. Our observations here are not intended to be universal for all game designs or designer aims, but instead are most relevant for designs where foreknowledge of the player's abilities is important. For our prototypes, being able to reason about the limits of player control is important, and we hope to explore the tradeoff of player freedom versus analysis tractability in the future.

Understanding

While procedural generators are commonly accepted as game components among many people who play games, controlling them so directly is something less often encountered. Most controllable generators are one-shot content generators which are configured once and then run at the start of a game (such as a world generator in Civilisation). To further complicate things, the control we allow the player in CLAY and Tombs involves directly influencing parameters normally expressed through code. Explaining what these are, why the player can control them, how they should do so and so on are all difficult to do, and we do not feel our initial attempts through CLAY were entirely successful. The theme of the game did not entirely explain to the player what process they were controlling, and the interface with which they interacted with the system was too complex.

While we are interested in improving player understanding and accessibility in the future, we are also interested in studying player experiences in games like Tombs. Such studies have obvious benefits in that they show whether the games are successful in explaining themselves to the player, but we are also interested in how the player's mental model of the procedural generator develops as they play the game. We believe that games like Tombs could become a tool for explaining generative systems to a user in an interactive way, by asking them to understand how generative space can be manipulated to achieve a goal.

CONCLUSION

In this paper we described two prototype games, CLAY and Tombs of Tomeria, in which the gameplay is driven by foregrounding a procedural content generator and allowing the player control over its parameters. We discussed the progression from our first prototype to our second, and outlined discussion points raised from the work that will pay into expanding these ideas in future.

BIBLIOGRAPHY

Mossmouth Games (2008) Spelunky [PC] Mossmouth Games.

Hello Games (2016) No Man's Sky [PC] Hello Games.

Mojang (2009) Minecraft [PC] Mojang.

Smith, G., Othenin-Girard, A., Whitehead, J., & Wardrip-Fruin, N. (2012, May). PCG-based game design: creating Endless Web. In Proceedings of the International Conference on the Foundations of Digital Games (pp. 188-195). ACM.

Treanor, M., Zook, A., Eladhari, M. P., Togelius, J., Smith, G., Cook, M., Thompson, T., Magerko, B., Levine, J. & Smith, A. (2015). Ai-based game design patterns. In Proceedings of the 10 International Conference on Foundations of Digital Games, FDG.